Занятие 6:
Основы многопоточности в Java

Что такое многопоточность и зачем она нужна?

Многопоточность - возможность разделить программу на два и более потоков, выполнение команд в которых не детерменировано по времени друг относительно друга. То есть код потоков выполняется одновременно и мы не можем иметь гарантию того, что одна команда одного потока выполнится раньше, или позже другой команды. Некоторые фрагменты могут получать детерменированность во времени друг относительно друга. Это называется синхронизацией потоков.

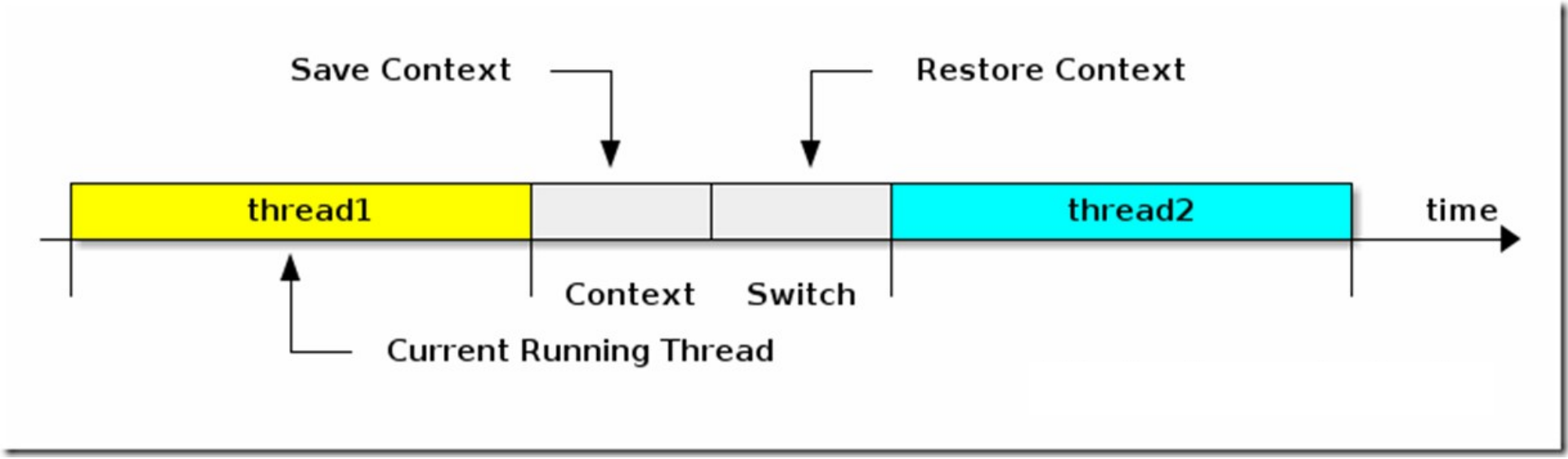
Преимущества:

- ✓ Параллельное выполнение нескольких процессов
- ✓ Perfomance
- ✓ Обслуживание нескольких пользователей
- ✓ Выведение операций, ожидающих реакции, в отдельный поток
- ✓ Выведение операций подготовки данных в отдельный поток
- ✓ Адаптация к «железу»

Вопрос: Сколько реальных отдельных потоков можно запустить в системе?

Псевдопараллельность

Псевдопараллельностью называют режим работы, в котором число свободных ядер меньше числа потоков приложения. В этом случае несколько потоков делят процессорное время между собой. При этом регулярно происходит переключение контекста



Создание потока в Java

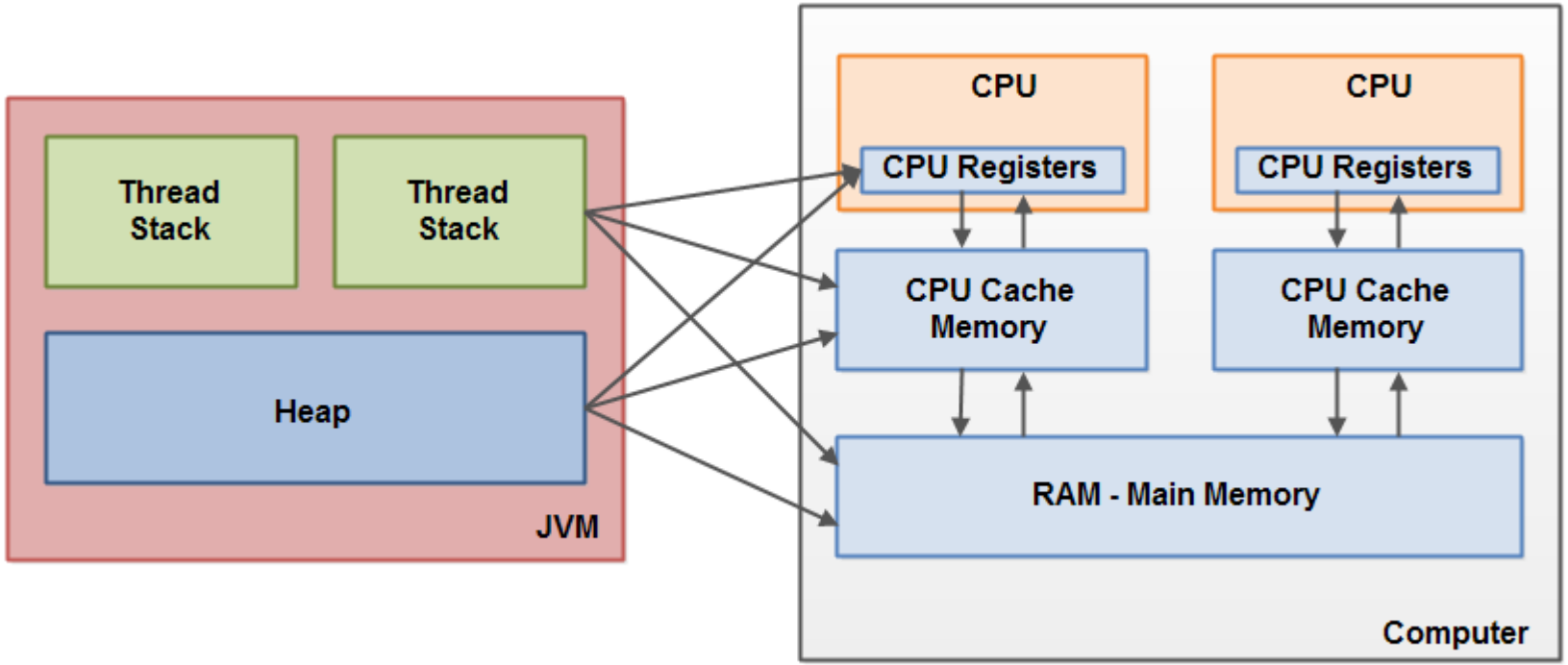
Способ 1

1. Создание класса-наследника Thread
2. Переопределение run()
3. Создание экземпляра класса
4. Вызов метода start();

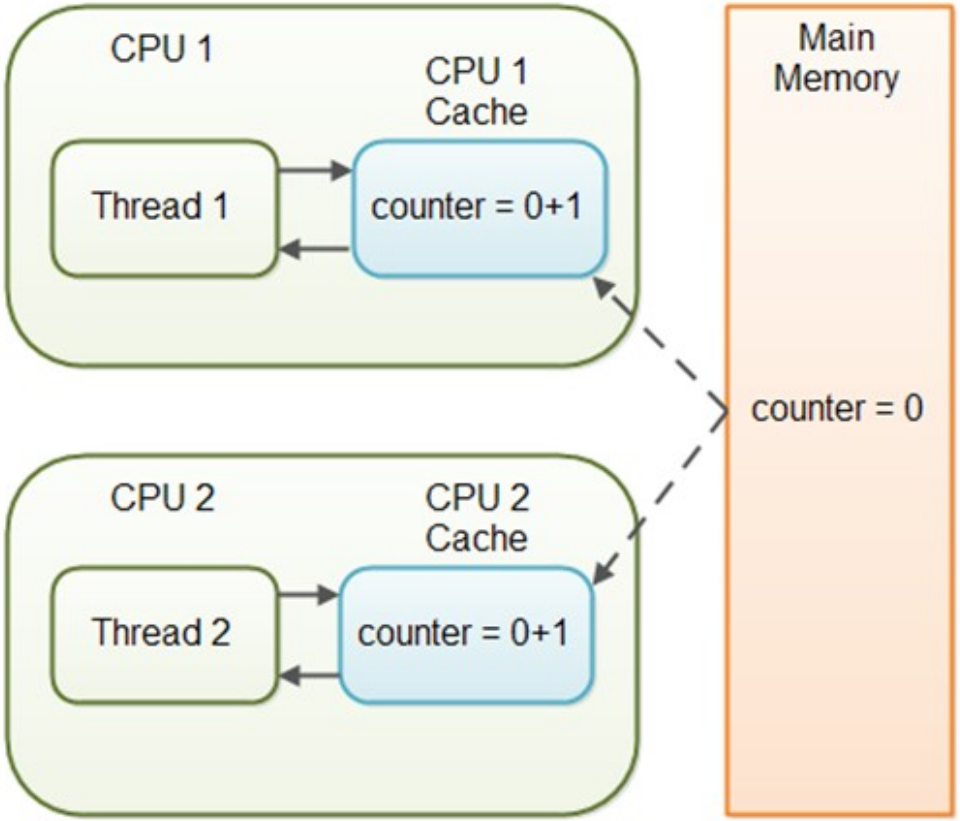
Способ 2

5. Создание класса, реализующего интерфейс Runnable
6. Реализация метода run();
7. Создание экземпляра написанного класса
8. Создание экземпляра класса Thread, получающего реализацию Runnable в качестве аргумента конструктора
9. Вызов start();

Стек потока


Thread Stack
Thread Stack
Heap
JVM
CPU
CPU Registers
CPU
CPU Registers
CPU Cache Memory
CPU Cache Memory
RAM - Main Memory
Computer

Борьба за ресурсы



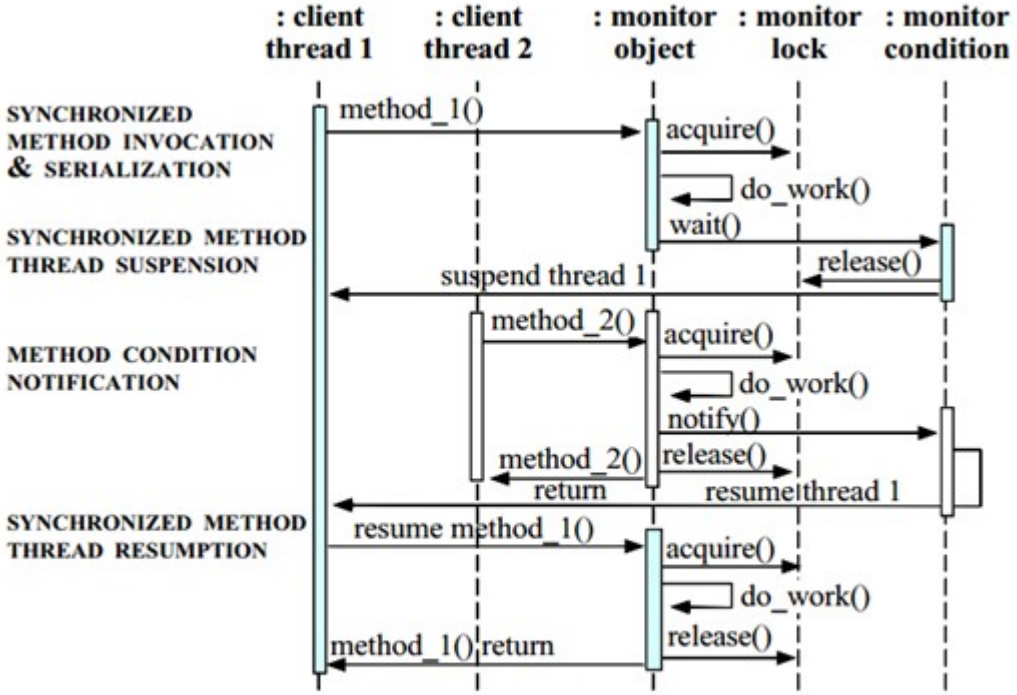
wait-notify-notifyAll

- ✓ wait(): освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify()
- ✓ notify(): продолжает работу потока, у которого ранее был вызван метод wait()
- ✓ notifyAll(): возобновляет работу всех потоков, у которых ранее был вызван метод wait()

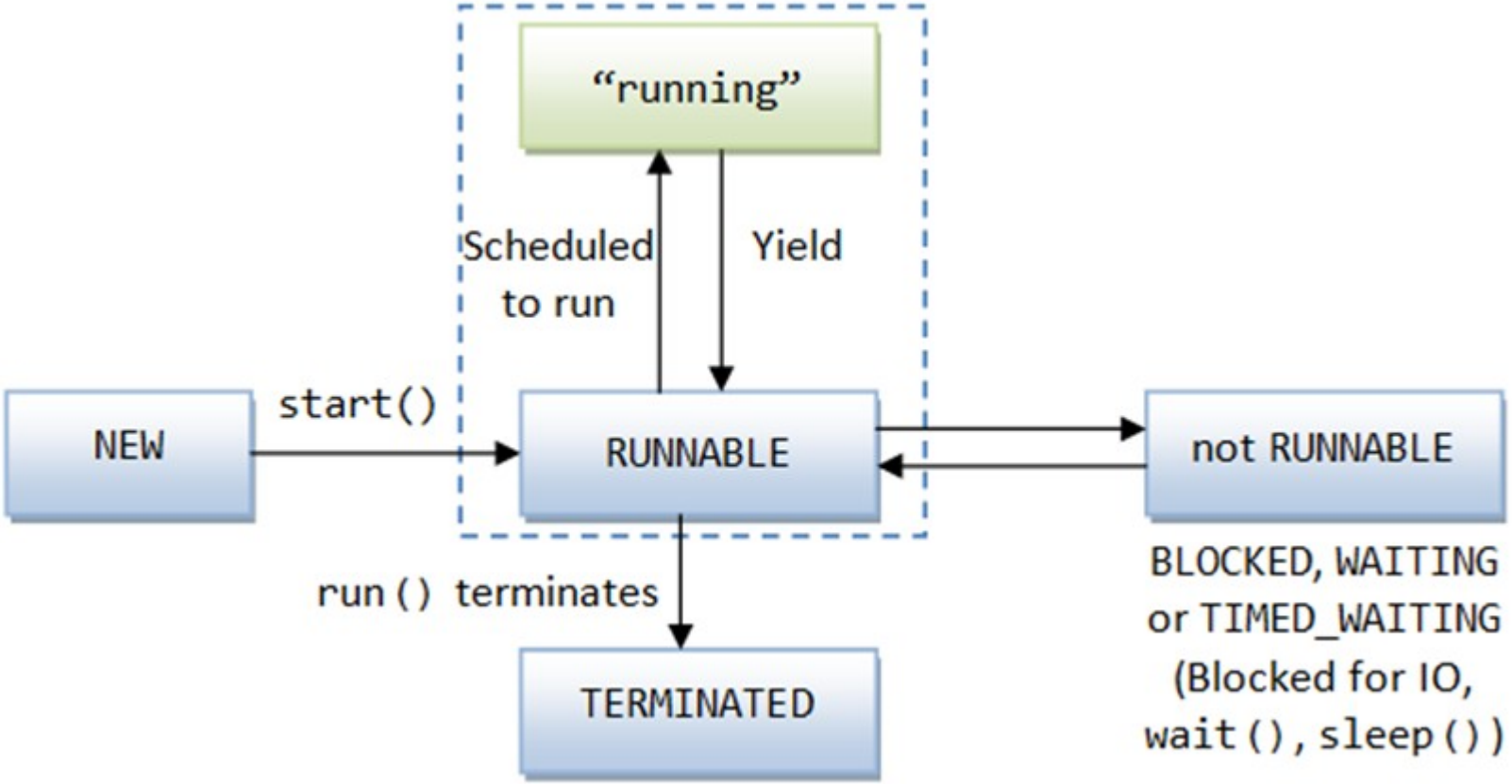
Остановка потока

Для остановки потока рекомендуется использовать безопасный метод
`Thread.interrupt()`
Он безопасен потому, что ничего не делает. Вызов метода не гарантирует нам НИ-ЧЕ-ГО!
Для контроля следует использовать проверку
`Thread.isInterrupted()`
и самостоятельно завершать поток.
Метод `join()` позволяет дождаться завершения другого потока

Synchronized-блоки



Состояния потока



Потоки-демоны

Потоки-демоны работают в фоновом режиме вместе с программой, но не являются неотъемлемой частью программы. Если какой-либо процесс может выполняться на фоне работы основных потоков выполнения и его деятельность заключается в обслуживании основных потоков приложения, то такой процесс может быть запущен как поток-демон. С помощью метода setDaemon(boolean value), вызванного вновь созданным потоком до его запуска, можно определить поток-демон.

Метод boolean isDaemon() позволяет определить, является ли указанный поток демоном или нет.

Что почитать

Кей С. Хорстманн, Гари Корнелл. **Java. Библиотека профессионала. Том 1. Основы**;
Стив Макконнелл. **Совершенный код**;
Брюс Эккель. **Философия Java**;
Герберт Шилдт. **Java 8: Полное руководство**;
Герберт Шилдт. **Java 8: Руководство для начинающих**.